HOG/linSVM 行人检测器的 OpenCV 实现 技术笔记

OnceMore2020

Abstract

HOG/linSVM 行人检测器在中等分辨率和低处理速度限制的应用场景下具有明显优势。笔记参考 [1], 记录了 HOG/linSVM 分类器涉及到的基本理论，并记录了使用 OpenCV 运算库中提供的 HOG 描述符实现 HOG/linSVM 分类器的方法。

Index Terms

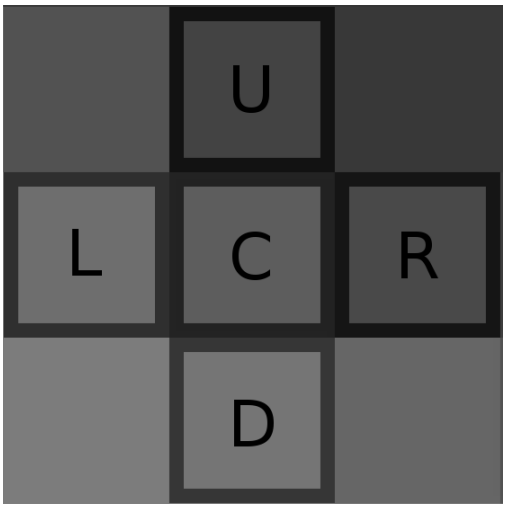
HOG 特征,linSVM, 分类器,OpenCV, 行人检测, 技术笔记.

I. 理论储备

主要参考 Dalal 和 Triggs 在 2005 年的 CVPR 会议上提出的 HOG 行人检测器，即 [1] 这篇“里程碑”式的论文, 以及 [3], [4] 的文档。

A. 梯度矢量

梯度矢量 (gradient vector) 是计算机视觉中的一个重要概念，许多视觉算法都需要引入对图像中每一个像素的梯度矢量的计算。如下图显示的 3×3 灰度图像，相应的像素标记字母作为标号。



像素值在 $0-255$ 之间，0 表示黑色，255 表示白色。$R-L$ 称为 x 方向变化率。需要注意的是，图中 L 像素的灰度值比 R 像素灰度值高，这样计算出来会是负值，$L-R$ 则是正值，也称为 x 方向变化率，但是一副图像中的计算方法应当保持一致。类似的，$U-D$ 称为 y 方向变化率。两个方向的变化率取值在 $-255 \rightarrow 255$ 之间，不能用一个字节存储，所以需要映射到 $0 \rightarrow 255$ 之间。如果将变化率用灰度值表示，则非常大的负变化率将映射为黑色，非常大的正变化率将映射为白色。同时，我们可以得到一个梯度矢量 $[R-L, U-D]$，其幅度 (magnitude) 和相角 (angle) 计算方法如下：

$$Magnitude = \sqrt{(R-L)^2 + (U-D)^2}$$

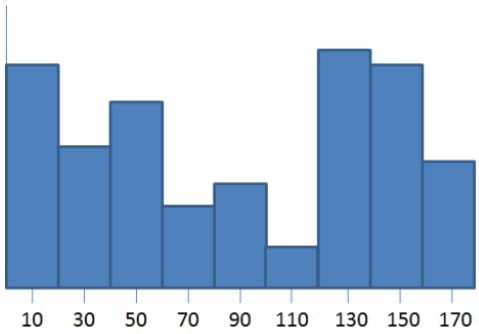
$$Angle = arctan\left(\frac{R-L}{U-D}\right)$$

梯度矢量很好地提取了边缘信息。另一方面，试想将图像的明亮度提升，即将图像中每个像素值加上同一个常数，重新计算梯度矢量会发现和明亮度变换之前的梯度矢量一致，这种性质使得梯度矢量可以被应用到特征提取中，即本文的人体特征提取中。

B. 方向梯度直方图

如下图的一个包含行人的图像，红色框标记一个 8×8 单元，这些 8×8 的单元将被用来计算 HOG 描述符。 在每

个单元中，我们在每个像素上计算梯度矢量，将得到 64 个梯度矢量，梯度矢量相角在 $0° \to 180°$ 之间分布，我们对相角进行分箱 (bin)，每箱 20°，一共 9 箱。具有某一相角的梯度矢量的幅度按照权重分配给直方图。例如，一个具有 85 度相角的梯度矢量将其幅度的 1/4 分配给中心为 70° 的箱，将剩余的 3/4 幅度分配给中心为 90° 的箱。这样就得到了下面的方向梯度直方图。



上面分配幅度的方法可以减少恰好位于两箱边界的梯度矢量的影响，否则，如果一个强梯度矢量恰好在边界上，其相角的一个很小的绕动都将对直方图造成非常大的影响。

另一方面，特征的复杂程度读分类器的影响很大。通过直方图的构造，我们将特征 64 个二元矢量量化为特征 9 个值，很好地压缩了特征的同时保留了单元的信息。设想对图像加上一些失真，方向梯度直方图的变化也不会很大，这是 HOG 特征的优点。同时，直方图分箱还有其他的方法 (本质上是量化)。

前面提到，对图像所有像素进行加减后梯度矢量不变，接下来引入梯度矢量的标准化，使得其在像素值进行乘法运算后仍然保持不变。如果对单元内的像素值都乘以某一常数，梯度矢量的幅度明显会发生变化，幅度会增加常数因子, 相角保持不变，这会造成整个直方图的每个箱的幅度增加常数因子。为了解决这个问题，需要引入梯度矢量标准化，将梯度矢量除以其幅度，梯度矢量的幅度将保持 1，但是其相角不会发生变化。引入梯度矢量标准化以后，直方图各箱幅度在图像像素值整体乘以某个因子 (变化对比度) 时不会发生变化。

除了对每个单元的直方图进行标准化外，另外一种方法是将固定数量的单元封装成区块，然后在区块上进行标准化。Dalal 和 Triggs 使用 2×2 区块 (50% 重叠)，即 16×16 象素。将一个区块内的四个单元的直方图信息整合为 36 个值的特征 (9×4), 然后对这个 36 元矢量进行标准化。标准化的算法不止上面提到的一种, [1] 中作者提供了 4 种标准化的方法。

区块重叠的影响是使得每个单元会在最终得到的 HOG 描述符中出现次数大于 1 次 (角单元出现 1 次，边单元出现 2 次，其它单元出现 4 次)，但每次出现都在不同的区块进行标准化。定义一个区块位移的步长为 8，则可以实现 50% 的重叠。

如果检测器窗口为 64 像素，则会被分为 7×15 区块，每个区块包括 2×2 个单元，每个单元包括 8×8 像素，每个区块进行 9 箱直方图统计 (36 值)，最后的总特征矢量将有 $7 \times 15 \times 4 \times 9 = 3780$ 个特征值元素。

II. OpenCV 实现

OpenCV 提供了采用 HOG 特征描述符实现的人体检测的例程：/samples/cpp/peopledetect.cpp。同时提供了 GPU 和 OPENCL 加速的 HOG 特征描述符，分别为 gpu::HOGDescriptor 和 ocl::HOGDescriptor。GPU 加速的实现：/samples/gpu/hog.cpp。OPENCL 加速的实现：/samples/ocl/hog.cpp。

OpenCV 提供的头文件在 include 文件夹中，在 Ubuntu 下使用源码编译 OpenCV 后可以在 /usr/local/include 文件夹下找到。

A. *CPU HOG* 简单例程

HOGDescriptor 类定义在 object.hpp 中，一个采用 HOGDescriptor 的简单实现 - /samples/cpp/peopledetect.cpp 代码如下：

```
1  #include "opencv2/imgproc/imgproc.hpp"
2  #include "opencv2/objdetect/objdetect.hpp"
3  #include "opencv2/highgui/highgui.hpp"
4
5  #include <stdio.h>
6  #include <string.h>
7  #include <ctype.h>
8
9  using namespace cv;
10 using namespace std;
11
12 // static void help()
13 // {
14 //     printf(
15 //             "\nDemonstrate the use of the HoG descriptor using\n"
16 //             "  HOGDescriptor::hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());\n"
17 //             "Usage:\n"
18 //             "./peopledetect (<image_filename> | <image_list>.txt)\n\n");
19 // }
20
21 int main(int argc, char** argv)
22 {
23     Mat img;
24     FILE* f = 0;
25     char _filename[1024];
26
27     if( argc == 1 )
28     {
29         printf("Usage: peopledetect (<image_filename> | <image_list>.txt)\n");
30         return 0;
31     }
32     img = imread(argv[1]);
33
34     if( img.data )
35     {
36         strcpy(_filename, argv[1]);
37     }
38     else
39     {
40         f = fopen(argv[1], "rt");
41         if(!f)
42         {
43             fprintf( stderr, "ERROR: the specified file could not be loaded\n");
44             return -1;
45         }
46     }
47
```

```
48      HOGDescriptor hog;
49      hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());
50      namedWindow("people␣detector", 1);
51
52      for(;;)
53      {
54          char* filename = _filename;
55          if(f)
56          {
57              if(!fgets(filename, (int)sizeof(_filename)-2, f))
58                  break;
59              //while(*filename && isspace(*filename))
60              //  ++filename;
61              if(filename[0] == '#')
62                  continue;
63              int l = (int)strlen(filename);
64              while(l > 0 && isspace(filename[l-1]))
65                  --l;
66              filename[l] = '\0';
67              img = imread(filename);
68          }
69          printf("%s:\n", filename);
70          if(!img.data)
71              continue;
72
73          fflush(stdout);
74          vector<Rect> found, found_filtered;
75          double t = (double)getTickCount();
76          // run the detector with default parameters. to get a higher hit-rate
77          // (and more false alarms, respectively), decrease the hitThreshold and
78          // groupThreshold (set groupThreshold to 0 to turn off the grouping completely).
79          hog.detectMultiScale(img, found, 0, Size(8,8), Size(32,32), 1.05, 2);
80          t = (double)getTickCount() - t;
81          printf("tdetection␣time␣=␣%gms\n", t*1000./cv::getTickFrequency());
82          size_t i, j;
83          for( i = 0; i < found.size(); i++ )
84          {
85              Rect r = found[i];
86              for( j = 0; j < found.size(); j++ )
87                  if( j != i && (r & found[j]) == r)
88                      break;
89              if( j == found.size() )
90                  found_filtered.push_back(r);
91          }
92          for( i = 0; i < found_filtered.size(); i++ )
93          {
94              Rect r = found_filtered[i];
95              // the HOG detector returns slightly larger rectangles than the real objects.
96              // so we slightly shrink the rectangles to get a nicer output.
97              r.x += cvRound(r.width*0.1);
98              r.width = cvRound(r.width*0.8);
99              r.y += cvRound(r.height*0.07);
100             r.height = cvRound(r.height*0.8);
101             rectangle(img, r.tl(), r.br(), cv::Scalar(0,255,0), 3);
102         }
103         imshow("people␣detector", img);
104         int c = waitKey(0) & 255;
105         if( c == 'q' || c == 'Q' || !f)
106             break;
107     }
108     if(f)
109         fclose(f);
110     return 0;
111 }
```

程序采用了 HOGDescriptor::getDefaultPeopleDetector() 来加载默认的行人检测器，接下来的问题是怎样使用手里已有的数据集来训练自己的分类器，这就需要了解 OpenCV 提供的 HOG 描述符的参数。

B. gpu::HOGDescriptor 参数

可以在 objdetect.hpp 中找到 HOGDescriptor 的类定义：

```
struct CV_EXPORTS_W HOGDescriptor
{
public:
    enum { L2Hys=0 };
    enum { DEFAULT_NLEVELS=64 };

    CV_WRAP HOGDescriptor() : winSize(64,128), blockSize(16,16), blockStride(8,8),
        cellSize(8,8), nbins(9), derivAperture(1), winSigma(-1),
        histogramNormType(HOGDescriptor::L2Hys), L2HysThreshold(0.2), gammaCorrection(true),
        nlevels(HOGDescriptor::DEFAULT_NLEVELS)
    {}

    CV_WRAP HOGDescriptor(Size _winSize, Size _blockSize, Size _blockStride,
                  Size _cellSize, int _nbins, int _derivAperture=1, double _winSigma=-1,
                  int _histogramNormType=HOGDescriptor::L2Hys,
                  double _L2HysThreshold=0.2, bool _gammaCorrection=false,
                  int _nlevels=HOGDescriptor::DEFAULT_NLEVELS)
    : winSize(_winSize), blockSize(_blockSize), blockStride(_blockStride), cellSize(_cellSize),
    nbins(_nbins), derivAperture(_derivAperture), winSigma(_winSigma),
    histogramNormType(_histogramNormType), L2HysThreshold(_L2HysThreshold),
    gammaCorrection(_gammaCorrection), nlevels(_nlevels)
    {}

    CV_WRAP HOGDescriptor(const String& filename)
    {
        load(filename);
    }

    HOGDescriptor(const HOGDescriptor& d)
    {
        d.copyTo(*this);
    }

    virtual ~HOGDescriptor() {}

    CV_WRAP size_t getDescriptorSize() const;
    CV_WRAP bool checkDetectorSize() const;
    CV_WRAP double getWinSigma() const;

    CV_WRAP virtual void setSVMDetector(InputArray _svmdetector);

    virtual bool read(FileNode& fn);
    virtual void write(FileStorage& fs, const String& objname) const;

    CV_WRAP virtual bool load(const String& filename, const String& objname=String());
    CV_WRAP virtual void save(const String& filename, const String& objname=String()) const;
    virtual void copyTo(HOGDescriptor& c) const;

    CV_WRAP virtual void compute(const Mat& img,
                         CV_OUT vector<float>& descriptors,
                         Size winStride=Size(), Size padding=Size(),
                         const vector<Point>& locations=vector<Point>()) const;
    //with found weights output
    CV_WRAP virtual void detect(const Mat& img, CV_OUT vector<Point>& foundLocations,
                        CV_OUT vector<double>& weights,
                        double hitThreshold=0, Size winStride=Size(),
                        Size padding=Size(),
                        const vector<Point>& searchLocations=vector<Point>()) const;
    //without found weights output
    virtual void detect(const Mat& img, CV_OUT vector<Point>& foundLocations,
                        double hitThreshold=0, Size winStride=Size(),
                        Size padding=Size(),
                        const vector<Point>& searchLocations=vector<Point>()) const;
    //with result weights output
    CV_WRAP virtual void detectMultiScale(const Mat& img, CV_OUT vector<Rect>& foundLocations,
                                  CV_OUT vector<double>& foundWeights, double hitThreshold=0,
                                  Size winStride=Size(), Size padding=Size(), double scale=1.05,
                                  double finalThreshold=2.0,bool useMeanshiftGrouping = false) const;
    //without found weights output
    virtual void detectMultiScale(const Mat& img, CV_OUT vector<Rect>& foundLocations,
                                  double hitThreshold=0, Size winStride=Size(),
```

```
                                Size padding=Size(), double scale=1.05,
                                double finalThreshold=2.0, bool useMeanshiftGrouping = false) const;

    CV_WRAP virtual void computeGradient(const Mat& img, CV_OUT Mat& grad, CV_OUT Mat& angleOfs,
                                Size paddingTL=Size(), Size paddingBR=Size()) const;

    CV_WRAP static vector<float> getDefaultPeopleDetector();
    CV_WRAP static vector<float> getDaimlerPeopleDetector();

    CV_PROP Size winSize;
    CV_PROP Size blockSize;
    CV_PROP Size blockStride;
    CV_PROP Size cellSize;
    CV_PROP int nbins;
    CV_PROP int derivAperture;
    CV_PROP double winSigma;
    CV_PROP int histogramNormType;
    CV_PROP double L2HysThreshold;
    CV_PROP bool gammaCorrection;
    CV_PROP vector<float> svmDetector;
    CV_PROP int nlevels;


   // evaluate specified ROI and return confidence value for each location
   void detectROI(const cv::Mat& img, const vector<cv::Point> &locations,
                                CV_OUT std::vector<cv::Point>& foundLocations, CV_OUT std::vector<double>& confidences,
                                double hitThreshold = 0, cv::Size winStride = Size(),
                                cv::Size padding = Size()) const;

   // evaluate specified ROI and return confidence value for each location in multiple scales
   void detectMultiScaleROI(const cv::Mat& img,
                                                   CV_OUT std::vector<cv::Rect>& foundLocations,
                                                   std::vector<DetectionROI>& locations,
                                                   double hitThreshold = 0,
                                                   int groupThreshold = 0) const;

   // read/parse Dalal's alt model file
   void readALTModel(std::string modelfile);
   void groupRectangles(vector<cv::Rect>& rectList, vector<double>& weights, int groupThreshold, double eps) const;
};
```

但是，OpenCV 文档里好像找不到对应的文档，只有 GPU 加速和 OCL 加速的 HOGDescriptor 的文档，例如，gpu::HOGDescriptor 的类实现：

```
struct CV_EXPORTS HOGDescriptor
{
    enum { DEFAULT_WIN_SIGMA = -1 };
    enum { DEFAULT_NLEVELS = 64 };
    enum { DESCR_FORMAT_ROW_BY_ROW, DESCR_FORMAT_COL_BY_COL };

    HOGDescriptor(Size win_size=Size(64, 128), Size block_size=Size(16, 16),
                  Size block_stride=Size(8, 8), Size cell_size=Size(8, 8),
                  int nbins=9, double win_sigma=DEFAULT_WIN_SIGMA,
                  double threshold_L2hys=0.2, bool gamma_correction=true,
                  int nlevels=DEFAULT_NLEVELS);

    size_t getDescriptorSize() const;
    size_t getBlockHistogramSize() const;

    void setSVMDetector(const vector<float>& detector);

    static vector<float> getDefaultPeopleDetector();
    static vector<float> getPeopleDetector48x96();
    static vector<float> getPeopleDetector64x128();

    void detect(const GpuMat& img, vector<Point>& found_locations,
                double hit_threshold=0, Size win_stride=Size(),
                Size padding=Size());

    void detectMultiScale(const GpuMat& img, vector<Rect>& found_locations,
                          double hit_threshold=0, Size win_stride=Size(),
                          Size padding=Size(), double scale0=1.05,
```

```
29                          int group_threshold=2);
30
31     void getDescriptors(const GpuMat& img, Size win_stride,
32                         GpuMat& descriptors,
33                         int descr_format=DESCR_FORMAT_COL_BY_COL);
34
35     Size win_size;
36     Size block_size;
37     Size block_stride;
38     Size cell_size;
39     int nbins;
40     double win_sigma;
41     double threshold_L2hys;
42     bool gamma_correction;
43     int nlevels;
44
45 private:
46     // Hidden
47 }
```

文档 [2] 中提到 (从类定义中也可以看出)，GPU 加速的 HOGDescriptor 类的接口和 CPU HOGDescriptor 尽量保持一致，那么下面结合 GPU::HOGDescriptor 的 API Reference 来了解其参数。

1) HOGDescriptor::HOGDescriptor: 创建 HOG 描述符和检测器。

函数原型：

HOGDescriptor::HOGDescriptor(Size win_size=Size(64, 128), Size block_size=Size(16, 16), Size block_stride=Size(8, 8), Size cell_size=Size(8, 8), int nbins=9, double win_sigma=DEFAULT_WIN_SIGMA, double threshold_L2hys=0.2, bool gamma_correction=true, int nlevels=DEFAULT_NLEVELS)

参数解释：

- **win_size** 检测器窗口尺寸，需要和区块尺寸和区块步进对齐。
- **block_size** 区块尺寸，需要和单元尺寸对齐。
- **block_stride** 区块步进，需要是单元尺寸的整数倍。
- **cell_size** 单元尺寸。
- **nbins** 分箱数量。
- **win_sigma** 高斯平滑窗口参数。
- **threshold_L2hys** L2-Hys 标准化参数 [5]。
- **gamma_correction** 标识是否要加入伽玛校正预处理模块。
- **nlevels** 检测窗口增量最值

2) HOGDescriptor::getDescriptorSize: 返回分类需要的系数数量。

3) HOGDescriptor::getBlockHistogramSize: 返回区块直方图的大小。

4) HOGDescriptor::setSVMDetector: 设置线性 SVM 分类器的参数。

5) HOGDescriptor::getDefaultPeopleDetector: 返回 OpenCV 提供的用于人体检测的分类器参数 (默认检测器窗口尺寸)

6) HOGDescriptor::getPeopleDetector48x96: 返回 OpenCV 提供的用于人体检测的分类器参数 (48×96 窗口尺寸)

7) HOGDescriptor::getPeopleDetector64x128: 返回 OpenCV 提供的用于人体检测的分类器参数 (64×128 窗口尺寸)

8) HOGDescriptor::detect: 进行对象检测，不进行检测窗口多尺寸缩放。

函数原型：

void gpu::HOGDescriptor::detect(const GpuMat& img, vector<Point>& found_locations, double hit_threshold=0, Size win_stride=Size(), Size padding=Size())

参数解释：

- **img** 图像源文件，目前支持 CV_8UC1 和 CV_8UC4 格式的图像。
- **found_locations** 检测到的目标对象边界左上角的位置。
- **hit_threshold** 特征和 SVM 分类判定平面之间的距离门限。
- **win_stride** 窗口步进，需要是区块步进的整数倍。

9) HOGDescriptor::detectMultiScale: 进行多尺寸窗口目标对象检测。

函数原型：

HOGDescriptor::detectMultiScale(const GpuMat& img, vector<Rect>& found_locations, double hit_threshold=0, Size win_stride=Size(), Size padding=Size(), double scale0=1.05, int group_threshold=2)

参数解释：

- **img** 同 HOGDescriptor::detect()
- **found_locations** 检测到的对象边界
- **hit_threshold** 同 HOGDescriptor::detect()
- **win_stride** 同 HOGDescriptor::detect()
- **padding** 同 HOGDescriptor::detect()
- **scale0** 检测窗口增量系数
- **group_threshold** 控制相似度门限，某些对象会被多个矩形覆盖，使用分组对这些矩形进行聚类。

10) HOGDescriptor::getDescriptors: 返回对整个图像计算得到的区块描述符。用于分类器的训练。

函数原型:

void gpu::HOGDescriptor::getDescriptors(const GpuMat& img, Size win_stride, GpuMat& descriptors, int descr_format=DESCR_FORMAT_COL_BY_COL)

参数解释：

- **img** 源文件
- **win_stride** 窗口步进，必须是区块步进的整数倍
- **descriptors** 二维数组描述符
- **descr_format** 描述符存储格式，有两种选择：
  DESCR_FORMAT_ROW_BY_ROW(行主序) 和 DESCR_FORMAT_COL_BY_COL(列主序)

C. GPU 加速 HOG/linSVM 程序实现

References

[1] N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” Proc. IEEE Int’ l Conf. Computer Vision and Pattern Recognition, pp. 886-893, 2005.
[2] Object Detection,http://docs.opencv.org/modules/gpu/doc/object_detection.html
[3] HOG PERSON DETECTOR TUTORIAL,http://chrisjmccormick.wordpress.com/2013/05/09/hog-person-detector-tutorial/
[4] GRADIENT VECTORS,http://chrisjmccormick.wordpress.com/2013/05/07/gradient-vectors/
[5] D.G.Lowe.Distinctive image features from scale-invariant keypoints.IJCV,2004.
[6] Histogram of oriented gradients,http://en.wikipedia.org/wiki/Histogram_of_oriented_gradients